WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

# IMPLEMENTING MESSAGE ORIENTED TRANSACTION PROCESSING FOR DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

Subhash Bhalla

Stuart E. Madnick

May 1988 #WP 2022-88

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

# IMPLEMENTING MESSAGE ORIENTED TRANSACTION PROCESSING FOR DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

Subhash Bhalla

Stuart E. Madnick

May 1988 #WP 2022-88

# Implementing Message Oriented Transaction Processing for Distributed Database Management Systems

Subhash Bhalla*, Stuart E. Madnick

Center for Information Systems Research
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA.

ABSTRACT

A flexible approach to solve several problems associated with transaction processing, in distributed database management systems, has been proposed in this paper. This approach facilitates concurrency control and recovery. Participating sites are permitted to lag behind in this scheme, thereby relaxing the time constraints and overheads traditionally imposed for handling multi-site updates and recovery from site-crashes. The proposed scheme is flexible to permit its adoption for a broad range of transaction processing environments.

KEY WORDS

Distributed database, Transaction processing, Synchronization, Recovery, Reliability, Site crashes, Time-critical transactions, Read-only transactions.

---

* On leave from School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi-110067, India.

5/20/89

# Implementing Message Oriented Transaction Processing for Distributed Database Management Systems

Subhash Bhalla, Stuart E. Madnick

## 1. INTRODUCTION

Transactions, as a concept incorporating atomic actions, are gaining increasing acceptance as an organizational unit of processing for a wide class of distributed applications. A number of approaches have been proposed for achieving efficient transaction processing [WIEH87] [GRAY81] [GRAY80] [LAMP81] [SPEC84] [ALLC83].

Existing approaches to transaction processing can be classified into two groups. First, in which isolated techniques are used to perform concurrency control, dead-lock detection, recovery, transaction commit, and replication control [GARC87] [CHAN87] [LIND87]. The second group uses integrated approaches to solving multiple problems within the same overall plan. Most schemes in this area maintain multiple versions of data items and are called multiversion techniques [REED83] [BAYE80] [CHAN85] [DUBO82] [STEA81] [VIEM82] [WEIH87].

Considering the existing approaches to transaction processing, a number of prototype systems such as distributed INGRES, SDD-1, R* and DDM, have demonstrated the feasibility of adopting isolated techniques for transaction processing [STON77] [ROTH80] [LIND87] [CHAN87]. At the same time, the implementations incur costly processing steps and communication overhead as each

individual technique attempts to adjust to constraints imposed by other processing steps [GARC87]. For example, in the case of R*, suppporting replicated or partitioned tables is a complex task [LIND87]. In general, supporting multi-site transaction commit and replicated copies, in the face of failures poses difficulties and requires further adoption of costly corrective steps [GRAY86].

Existing trends in the area of transaction processing in Distributed Database Management environment, point to the following desirable features:

(a) Read-only transactions (queries) and updates should not interfere, so that, queries can proceed without interference or roll-backs, and access the most recent consistent version of data.

(b) On-line dumping of the database, for recovery from media failures, is faciliated.

(c) Adequate level replication is supported with little or no losses, in terms of response degradation.

(d) Update transaction recovery and system recovery are simplified.

In the next section we discuss some key issues connected with message oriented transaction processing. Section 3 describes the proposed transaction processing technique. Section 4 discuses problems related to site failures and recovery. Section 5 is

devoted to imlementation considerations. The last two sections are devoted to considerations of performance comparison and discussion.

## 2. TRANSFORMATION OF DATABASE STATE

Each transaction that updates objects in a database transforms the database to a new state. The system state includes assertions about values of records and about the allowed transformations of the values. Transactions obey the assertions (laws of consistency constraints) by transforming consistent states into new consistent states.

Gray [GRAY81] points out a few desirable features of transaction processing systems that are not currently supported by such systems. He recommends the use of techniques such as time-domain addressing and also keeping a log of transaction processing activities (also see [ESWA76]). Recently, Gray [GRAY86] has suggested the idea of queued transaction processing (message oriented transaction processing) for decoupling the processing of the transaction from its submission and from the delivery of the response.

Earlier trends in this area were focused on multiversion techniques in which objects are never altered, rather an object is considered to have a time history. The object addresses become <name, time> rather than simply <name> [REED78]. In these

systems, each transaction is assigned a unique time of execution and all of its reads and writes are interpreted with respect to that time. More recently, Weihl [WEIH83] has proposed a strategy involving implemention of hybrid atomicity, for multiversion techniques. It favors use of time-stamps for updates as they are committed rather than alloting time-stamps before the transactions start executing. Another study [PAPA84], points to a close connection between reliability and multiversion techniques and mentions that, delaying the write steps until commit time, as adopted by conventional techniques, is a multiversion strategy similar to [REED78] [STEA81]. Similarly, Attar, et al., [ATTA84] have indicated a preference for timestamp based or multiversion based concurrency control techniques.

Recent trends in transaction processing, indicate increasing utilization of the history of transaction processing activity as an aid for transaction management. For example, [CHAN87] uses a list of committed transactions to facilitate processing of Read-only transactions. Similarly, [SARI86] uses update history information in order to process updates that arrive out of sequence. Most of the algorithms for recovery from network partitioning rely on conflict graphs created with the help of historical information pertaining to the transaction processing activity [DAVI85].

Based on the above ideas, we propose an implementation of a

transaction processing scheme that performs concurrency control using a history of committed transactions. For this, we have chosen Kung's [KUNG81] original proposal as the central scheme for synchronization. A list of committed transactions is generated as part of the process of performing concurrency control. The same pool of information is later used to perform other activities associated with transaction processing. Kung's proposal has been suitably modified to meet the requirements of transaction processing as discussed above.

In our approach, there is a central coordinator that validates all requests for transaction updates and generates a list of committed transactions. The idea of a coordinator is a logical one and rigid only to the extent that, at any time, one of the sites must lead. If the coordinator fails, another node (site) must be elected to replace it, as in [GARC82] [KIM 82] [DOLE82]. The common pool of information is maintained by all the participating nodes, and this enables the sites to operate in an autonomous manner.

## 3. PROPOSED APPROACH

In this section we provide steps for implementation of a transaction processing system based on the ideas of queued transactions and message based activities that aim at removing some of the defeciencies associated with current systems. Our

main objective is to show the feasibility of such an approach for a distributed database management system. The proposed technique is capable of supporting higher levels of replication of data with moderate volume of update activity, as the read-only transactions do not interfere with the update transactions. The various aspects of the proposed scheme are discussed below.

## 3.1 TRANSACTION PROCESSING

In the proposed approach all participating sites receive transaction requests. Each request is allotted an unique identification number (Transaction-id) and is processed further. The transaction executes in two phases [KUNG81]. During the first phase data items are read and transaction execution is completed. The final computed values for data items to be updated in the database are held in temporary storage. The second phase is the validation phase. During this phase a certification request is sent to the site that is currently acting as the coordinator.

The coordinator-site performs a validation check on the basis of a list of committed transactions which is generated by the coordinator and is maintained by all sites. At the end, it broadcasts a message in order to commit or reject the transaction. A transaction is considered to be committed, if another site (requesting site) receives the update message. The coordinator updates its CTL after performing (sending/

acknowledging ) the transaction update broadcast. In case of a reject message, the requesting site performs the transaction again. In case the transaction is committed, all participating sites receive the committed transaction entry and incorporate it in their list of committed transactions. The sites also receive the new update values for updating local copies. A benefit of this algorithm is that no validation overhead is incurred for read-only transactions. This is useful for applications where retrieval requests dominate update requests.

### 3.2 INFORMATION MANAGEMENT

A Transaction-id is assigned to each transaction on its arrival. It contains information related to site-of-origin, site-status, transaction type (read-only class/ duration class/ priority class; (if known)), and local sequence number. The site-status value indicates the Commit Sequence Number (CSN) of the most recent transaction broadcast received and implemented in the local database copy. Each participating site maintains this value. It is also noted along with read accesses made during transaction processing. It is used at the time of transaction validation to verify that data items read for transaction computation have the most recent value in the case update transactions.

The coordinator and the other sites maintain a table of

entries of all recently certified update transactions in a Committed Transaction List (CTL) as shown in Figure 1. This list contains entries for:

(a) Transaction-id;

(b) Read-set and associated details such as site at which access was made and the status of the site at the time of the Read-access;

(c) Write set; and

(d) Allotted Commit Sequence Number (CSN). These numbers are assigned in ascending order, i.e., Next CSN value = Previous CSN value + 1. The CSN also indicates the identity of the coordinator-site.

The master list is generated by the coordinator-site. Each of the other participating sites maintains a copy of the CTL, which is updated on the basis of update broadcasts received by it from the coordinator-site.

In addition to the above CTL, the sites also maintain a current Network Status Table, which contains information about the status of each participating site (Figure 2). This table facilitates recovery from site failures and is used to trim the size of CTL. The entries in this table are updated on the basis of regular messages received from the participating sites. Each entry in this table contains :

(a) The identifier of the site, Site-id;

| Allotted Commit Sequence Numbr (CSN) | Transaction-id | Read-set | Write-set |
|---|---|---|---|
| -- | -- | -- | -- |

Figure 1: Committed transactions list (CTL).

| Site Identification | Site-status in terms of most recent CSN known | Oldest Read-access status | Up/down status | Coordinator /not a Coordinator |
|---|---|---|---|---|
| -- | -- | -- | -- | -- |

Figure 2: Network status table.

| Transaction-id | Items Read | Version (site-status) | local access/ access by other sites |
|---|---|---|---|
| -- | -- | -- | -- |

1. Ongoing transactions at the local site, other than known read-only transactions
2. Accesses made by update transactions from other sites (not yet committed), other than known read-only transactions

Figure 3: List of read accesses made at the local site.

(b) The site-status value;

(c) Read Access Status, which indicates the oldest read-access by the executing transactions (the site-status of the accessed site, at the time of such an access);

(d) Up/down status; and

(e) Central site status (acting as coordinator site : yes or no).

The sites also keep a list of read accesses made at the local copy, by the executing transactions that have not yet been committed (Figure 3). This table is used for computing a value required for trimming the size of CTL and to know about transactions that are yet to commit as explained in the next section.

### 3.3 PROCESSING AT THE COORDINATOR-SITE

One of the sites is elected as the coordinator at the initial stage. This site receives update transactions for certification. All validated transactions are assigned a Commit Sequence Number (CSN), and transaction update messages are sent to the participating sites. The coordinator also performs other functions, which are described later in this section.

On receiving a request, the coordinator looks at the Read-set and the associated site-status values. All the entries (later than the site-status indicated with a read access) in the CTL are

checked for a conflict (see [REDD82b]). In case no conflict exists, the requesting site is informed of the decision and a new entry is added to the CTL. The next CSN value is assigned to the transaction and the update message is broadcasted to all sites to enable them to update their CTL copy.

Consider T as the set of committed transactions. A transaction t arrives with a site-status value "ss", as the status of the site, where the read access for performing transaction t have been made. The validation check and the subsequent tasks performed by the central-site can be algorithmically expressed as indicated below.

```
valid := true;
for  CSN from (ss+1) to latest CSN
do
if  (write-set of transaction T intersects with
     read-set of t)
then  valid := false; conflicting-csn := CSN;
end;
if  valid then
     next CSN = latest CSN+1
     broadcast t with next CSN, commit t;
else  reject t, communicate conflicting-csn to the
      requesting site;
end;
```

For all validated transactions, the update values for the items in the write-set are also sent to the participating sites. However, for the transactions that update large portions of the database, the update values need not be broadcasted by the coordinator. Instead, these may be sent directly by the requesting site to the other sites, after the transaction has been certified as valid. In the case of a reject message, the coordinator also communicates the identity of the most recent transaction that is in conflict with the requested update transaction.

In addition, critical changes to information related to the Network Status Table, such as recent site crashes, and read-access status changes are communicated by the central site to all participating sites. The participating sites are designed to maintain the network status table for greater reliability.

The CTL maintained at all sites will gradually grow in size. To trim its size, the following strategy can be adopted:

(a) If all the participating sites are operational and all of them contain a site-status higher than a specific value (say n), then all transaction entries with CSN < n can be removed from the CTL. The value of n can be obtained from the Network status table, as n = Minimum of (site-status values), considering all participating sites.

(b) The subset of transactions which are currently being processed and which have accessed sites at a time when the site had a site status value less than n, can be rejected and made to roll-back. Alternatively, considering the transactions which are currently being executed and the earliest read-access made by them. It is possible to determine a Base Sequence Value ( say "m" ). All entries in CTL with CSN smaller than this value can be deleted. The value of m can be obtained from the Network status table. m = Minimum of (oldest read-access status values), considering all participating sites.

A low value based on the computed values of "m" and "n" can be used to trim the size of the CTL. As all sites that maintain the CTL also maintain the Network status table, the task of reducing the size of the CTL, can be performed by individual sites on their own. The trimmed values can be dumped on a permanent storage media for future recovery requirements that may arise. This dumping operation may not be required at all sites (Figure 4).

In a fast network, such as an ETHERNET based LAN, communication and transaction certification at the coordinator site require a small duration of time. In such a case, the size of the CTL is likely to be small, unless :

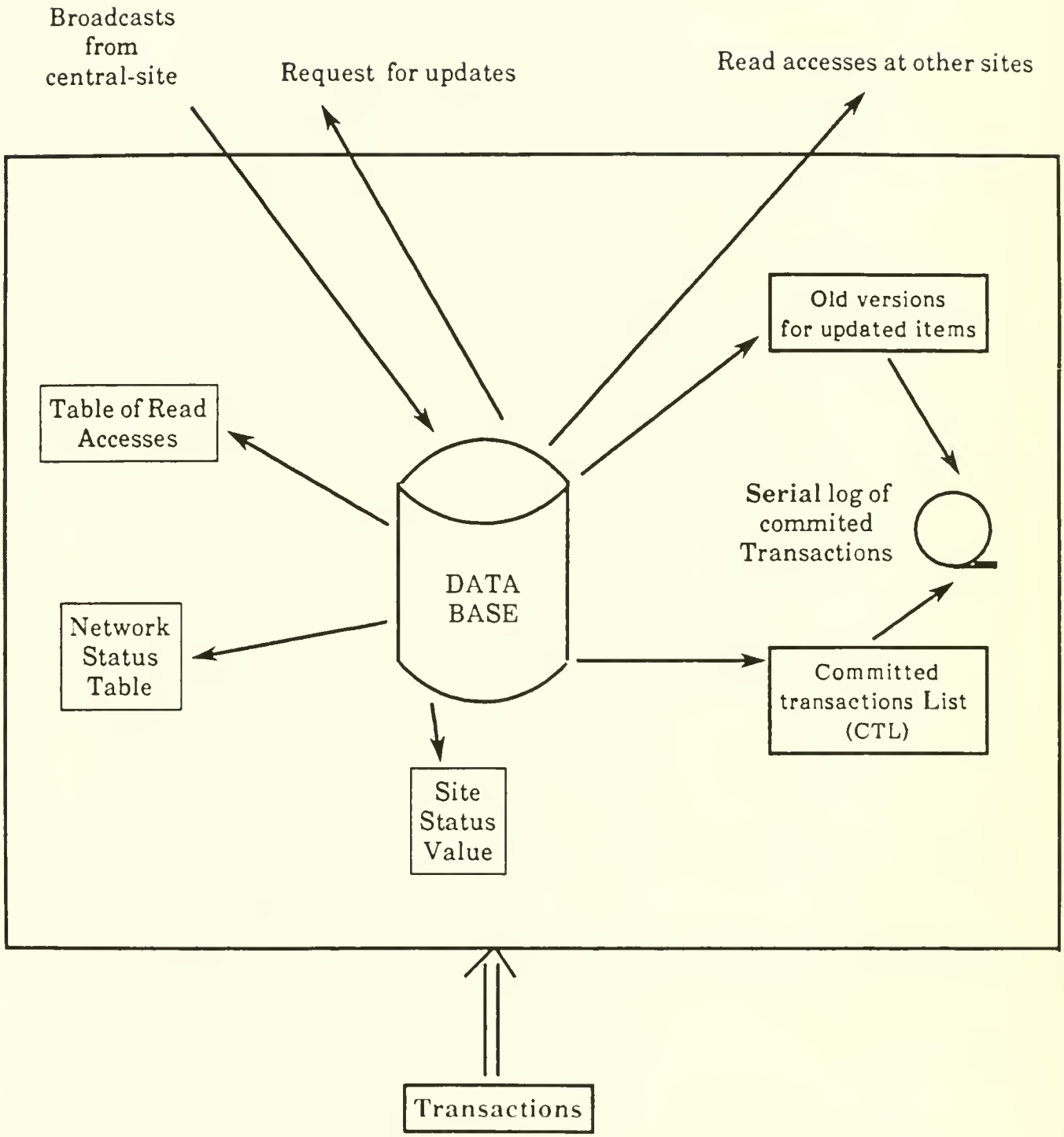


Figure 4: Transaction processing at a site

(i) one or more sites are not operational and hence the trimming cannot be done; or

(ii) transactions are taking a long time to be completed and hence a large portion of the CTL is required to perform certification for them.

Methods for dealing with these two situations are presented later in this paper. However, in all other cases, the size of the CTL will depend on the average time elapse required to perform the validation, the associated intersite communication, and the propogation of the updated values. It is likely to be limited to 2 to 3 hundred entries in a high volume transaction processing environment.

### 3.4 LOCAL PROCESSING

In addition to maintaining the two tables mentioned above, each participating site updates the local copy of the database and modifies the site-status value as per the CSN of each update message. The process of updating the local copy of the database is considered to be atomic.

The site-status at the participating sites may lag behind the central sites by a few broadcast messages at the time a transaction completes its computations. Before sending a request for update to the central site, the local site performs a local certification to ensure that the most recently communicated

transaction updates are not in conflict with the present transaction. It then prepares the update request and forwards the request to the coordinator-site (Figure 5). The certification check at the central site is a short check as recommended by Bernstein [BERN81] [REDD82a].

Apart from the above, a log of updated values can be maintained. As each update is implemented in the local copy of the database, a sequential log can be maintained indicating: old-values, new-values, transaction-id, and the Commit Sequence Number (CSN). Since all messages are delivered in the order in which they have been transmitted, the transaction updates occur in the sequence of the CSN values assigned to the transactions. All copies of this log will be identical to each other. This log permits swift recovery from failures caused by site crashes or network partitioning and enables processing of Read-only transactions requiring access to an older version (see [CHAN85]). No overhead is assumed for maintaining the log, since all database updates need to be implemented in an atomic manner through appropriate buffering and a single write statement.

As mentioned earlier, the local site receives transactions from the users. Each request is allotted a Transaction-id at the time of its arrival. The transaction is then allowed to proceed, irrespective of other on-going transactions. At the time of a read-access, the site-status of the accessed site is noted by the

(1) Transaction-id

(2) Read-set

| Site-id | Site-status | Items-read |
|---|---|---|
| -- | -- | -- |

(3) Write-set

| List of Items | Update values for broadcast |
|---|---|

(4) Oldest Read access status at the requesting site

Figure 5: Update request message from the requesting site.

transaction. The local site also keeps a table of accesses made by ongoing transactions (Figure 3). This table is used to efficiently compute the value $m_{\%}$ (base sequence value for an individual site, refered earlier). When processing of a transaction is initiated, the local site knows the sites that need to be accessed and the current (or recent) status of the sites to be accessed through its network status table enteries. It also checks entries in the table of accesses. The local site computes a value (say $m_{\%}$) for transaction i, such that all accesses made by the current transaction encounter site-status values higher than $m_{\%}$. The minimum of $m_{\%}$, considering all on-going transactions at a particular site, is notified to the central site along with regular update messages. The coordinator site communicates this value to the other sites along with the update message. It is used by the participating sites for updating enteries in the Network status table.

After performing the transaction the local site checks for one of the three classes into which the transaction may fall:

(a) read-only transaction;

(b) local update transaction; and

(c) global update transaction.

Transactions of class (a) and (b) can be certified locally. For transactions that are known to be of read-only type at the begining (based on their transaction-id), special computation

procedure may be adopted, as explained below.

### 3.4.1 READ-ONLY TRANSACTIONS

A transaction may be a read-only type of transaction based on its transaction-id. Participating sites do not record read accesses made by such transactions in their list of read accesses. Other transactions that do not request updates at the end of computation phase are also classified as read-only transactions for the purpose of further processing or processing after a restart. The termination of such a transaction must be notified to the accessed sites so that read access enteries within the list of read accesses at those sites can be deleted. A read-only transaction can be processed in multiple ways within the proposed system for transaction processing.

One of the ways is to process the transaction based on the idea proposed by Chan [CHAN85] for unblocked processing of distributed read-only transactions. The transactions access the most recent consistent version of data based on their transaction-id (includes the site-status of its site of origin) and old versions for updated items (Figure 4), being maintained by the participating sites. This process is most suitable for long transactions that access data that is not available locally, and transactions that do not critically depend on the most recent version of data. Alternatively, if after processing a read-only

transaction, it is realized on performing a local validation check that certain values are obsolete, then the appropriate old version of such items can be read to complete the transaction.

A second approach could be adopted for processing transactions that access locally available data. In order to avoid rollback the transaction may resort to blocking at the local site (discussed below under 'Blocking', for stalling such updates that update values read by the ongoing transaction). A large replication of data specifically for the convenience of read-only transactions may serve as a useful means of reducing delay due to intersite communication.

Another approach can be adopted to process transactions that need to make use of the most recent database state. Such transactions can be sent to the central site and processed in a manner identical to the global update transaction (may even resort to blocking at the central site if required in case of a rollback).

Alternative approaches may be used to suit the needs of a given application environment.

### 3.4.2 LOCAL UPDATE TRANSACTIONS

A local update transaction can be identified on the basis of the following criteria:

(a) All data being updated by the transaction is locally

available;

(b) No part of the data being updated is replicated elsewhere, for any purpose other than for processing read-only transactions at those sites; and

(c) No other ongoing transaction from another site is permitted to read any of the items being updated by the transaction.

A local update transaction is locally certified and committed with a sequence number based on the current site status, in case no replication exists. In the case of replicated copies, an update message need to be sent to each copy.

### 3.4.3 GLOBAL UPDATE TRANSACTIONS

All the other transactions are classified as the global update transactions. Such transactions also undergo a local certification before the update request is forwarded to the coordinator site. The local certification check ensures that the most recent changes to the database state, (received by the local site) are in no conflict with the current request. Next, the local site prepares and sends a message to the central-site (Figure 5). The central-site performs global certification and broadcasts a message, in the manner described earlier. On receiving a communication from the coordinator-site, the requesting site determines whether the transaction has been

committed or rejected. The reject message from the central site can also be designed to indicate the identity of transactions that are in conflict. The requesting site must ensure that it accesses the current version of the desired data items during the restarted computation phase.

By increasing the level of replication at the sites of the network, the time required to access sites and to process transactions can be reduced. The coordinator site performs certification on a centralized basis. This technique can be conveniently implemented in a broadcast based computer network. The overhead involved in other cases can be reduced by relaxing the reliability requirements or by modifying the algorithm.

## 3.5 PRIORITY RESPONSE AT A SITE

Time critical transactions such as in real-time applications are characterized by a fixed response-time constraint imposed on them. Within the proposed framework of transaction processing, a site can monitor a critical application. It can be considered to be using a segment of the database. For such a segment it can be permitted to perform update transactions as local updates with other sites maintaining copies of the database segment for processing read-only transactions. The site can perform a local update for a critical transaction and communicate the update message to the central site for broadcast.

Global update requests generated by other sites, that need to update contents of the critical database segment can submit the same to the site processing critical transactions. Alternatively, such transactions can be processed normally with the exception that the central site after validating the transaction, submits the same to the priority site that may accept or reject the request through a message. Based on the response the central site may further process the transaction. In such a case, transaction requests for blocking items that form part of the critical database segment will be ignored by the priority site. It is likely that some of these transactions may have to go through repeated roll-backs.

## 3.6 BLOCKING

In case a large number of update transactions access the same set of items, it is possible that a particular transaction may need to be roll-backed many times before it can be committed. Frequent roll-backs can degrade system performance. This problem can be solved by blocking requests. This concept is also relevent in other cases, such as:

(a) For processing time-critical transactions arriving at the node;

(b) For reorganizing of data dictionary; and

(c) For processing long transactions involving commonly

accessed items.

The technique of blocking is compatible with our design approach as described below.

On receiving a reject message, the requesting site makes a request to put items that it accessed in the blocked list. In the case of transactions that execute over a short duration of time, all the items that need to be accessed must be placed in the blocked items list. For processing long transactions, it may be desirable to send request for blocking items as and when required. In such a case, the co-ordinator site adds a dummy entry to the list of committed transactions by placing these items in the list with a dummy number as read-set and as write-set of a dummy transaction. It also broadcasts this entry for inclusion in the CTLs being maintained by the other sites. The requesting site proceeds to perform the transaction after ensuring that it accesses current database state.

The activity pertaining to the blocked items is stalled at the other local sites for a fixed time-out period so that the requesting site can complete the transaction. In the case of short transactions, serializing transactions in this manner (similar to locking) can be considered to be an appropriate solution, as multiple number of concurrent update transactions cannot have access to the same set of items. In the case of the long transactions such as batch-jobs the system should not

support more than one (per database segment) transaction at a given time. Such a restriction can prevent repeated roll-backs and the possibility of dead-locks. When a short transaction requires items blocked by a long transaction, the long transaction must either be completed or partially rolled back. Conflicts due to multiple short transactions accessing the same set of items are resolved locally by the sites. Such transactions are serialized by the coordinator and these execute in such a manner that dependent transactions wait for the completion of others ahead of them, in the serial order of execution. Short transactions must predeclare the items desired and request blocking for all items.

Based on the previous transaction computation time estimate, it is possible to compute an appropriate time-out period for the transactions that request blocking items. The requesting site is expected to complete the current transaction and to broadcast the write-set and updated values within this period. The participating sites can be designed to automatically remove the blocking entry on receipt of the update message for the corresponding transaction or on the expiry of the time-out period. In case the requesting site fails the item blocking entry may be deleted by all participating sites. This procedure will eliminate the occurrence of repeated roll-backs.

## 3.7 SITE CRASHES

Unlike most existing algorithms, our approach involves assignment of a unique identification to each committed transaction. Also, it permits maintenance of a site-status, in terms of transaction updates that have been completed. Since participating sites function independently of each other, they will frequently lag behind other sites, though on a very temporary basis, in terms of implementing updates. This asynchrony significantly reduces overheads involved in transaction commit and recovery procedures.

In case a participating site fails, the rest of the sites (including the coordinator site) may not be able to trim the size of the list of committed transactions, as one of the sites is not able to implement updates. In case the faulty site recovers soon enough, it can begin to paticipate normally. However, it may contain a low site status value and hence, all transactions processed by it will be checked against a long list of committed transactions. Alternatively, it must initiate activity to update itself from other sites in the network. Since all sites maintain a log of updated values (and old values) along with the list of committed transactions, a recovering site can seek the assistance of any of its peers to bring about its own recovery.

In case the faulty site does not start functioning within a predefined amount of time, it is suggested that the other sites

dump the list of committed transactions and the log of updated values to permanent storage. The failed site, on its recovery, can be supplied information from such a source. (This process is described in the succeeding section entitled "Site Initiation and Backup").

The coordinator site can also fail. The recovery process is slightly different in this case. Since all actions performed by the coordinator site are assumed to be atomic, it may fail either before broadcasting a validated transaction or after communicating the update message to one of the sites. On detecting coordinator node failure, the participating sites elect a new coordinator, in the manner suggested in [GARC82] or alternatively, on the basis of previously agreed dynamic succession list [KIM 82]. Also, Dolev and Strong [DOLE82] have proposed an authenticated version of the Byzantine consensus protocol for ensuring agreement in case of processor or communication failures. This protocol can be adopted to update network status table in case of a failure. In addition to the normal steps for electing a new coordinator, the sites will also need to ensure that all CTL entries (broadcast so far) are made available to the new coordinator site.

When the previous coordinator site recovers, it functions like other sites, without bearing the coordinator role.

## 3.8 COMMUNICATION FAILURES

Communication line failures are difficult to distinguish from site failures. In case, a single site cannot communicate with rest of the sites, it changes its network-status table entries. Assuming a large amount of replication, it can process read-only as well as local update transactions. Limited global update service is possible subject to the constraint that transactions can be rolled back or applied corrective strategy, if required [GARC81]. The site assumes the roll of the coordinator, while it is out of communication with other sites. On recovery of the communication line, the affected site will initiate a recovery procedure to communicate with the coordinator site. The coordinator site and the recovery site will need to merge their CTLs. Each committed transaction entry from the CTL of the recovering site needs to be validated by the coordinator site. On its recovery, the recovered site accepts the coordinator site as the coordinator and both update their network-status tables. The same strategy can be applied to recovery from network-partitioning, where two coordinators from two network partitions may merge their CTLs. More complex failures such as multiple site crashes need further consideration and are not being examined as a part of this study.

## 4. SITE INITIALIZATION AND BACKUP

Site initiation relates to the integration of a new site into an operational DDBMS [ATTA84]. In our approach, when a new site joins the DDBMS, a site-status value must be assigned to it, and the presence of the new site must be communicated to other sites.

In order to copy the database from another site, the new site notes its node-status and copies the database disregarding any on-going change to the state of the database. After completing a non-stop copying procedure, it is essential to note the changes to the state of the database through the extensions to the list of committed transactions. These extensions in the list can be used to overwrite the data items included in that list, in order to obtain a consistent copy of the database along with the corresponding site-status value. This procedure may need to be iterated more than once (two to three times), in order to achieve a recent site-status value at the site. Alternative approaches need to be examined in order to determine an optimum backup strategy.

Site backup involves creation of a static backup copy of the database for archival or query purposes [ATTA84]. One of the options is to make a new site join the network in the manner described above, and let it function for a sufficient amount of time, so that a backup copy is generated. A second option is to

initiate a read-only transaction which takes into account the site-status at its initiation and ignores all later changes to the database state (see discussion on read-only transactions). In the latter option, if the backup generation takes a long time, it will lead to a large CTL size.

There is another option. The participating sites perform periodic trim-down operations to reduce the size of the list of committed transactions. These operations can be used to generate a list of committed transactions along with a list of the changes to the state of the database. The lists can be recorded on permanent media for the participating database, at the time of trim-down. By merging these lists, and using them in conjunction with previous backup copies, more recent copies of database state can be generated.

## 5. IMPLEMENTATION CONSIDERATIONS

It is desirable to consider techniques for reducing the level of dependance on the central coordinator. One method is derived from the conventional token passing [LeLA78] [BERN81] approach. A cyclic ring of participating nodes is formed, with each node broadcasting one or all of its pending updates, and passing the token along with its node status and recent update information to the next node. For example, in case the network consists of a few nodes, each node can potentially serve as a

coordinator through a cyclic token passing scheme. In the case of a network with a large number of participating sites, a few selected nodes can be designated to form a pool of coordinator nodes which, in turn, may pass tokens among themselves to perform the function (also see, [SCHN79] and [MOHA82]). As all participating sites maintain the summary information required for validating transactions by entering the critical section, solutions such as those proposed by Ricart and Agarwala [RICA81] can be adopted for designing a decentralized solution. Such a solution is feasible for environments with large percentage of read-only transactions, and small number of participating sites.

The second approach is to consider use of a time-stamp based concurrency control algorithm, in which each update is allotted a unique identification number before it is communicated to other sites. The multi-site update and recovery processess can still be handled in the same fashion, with each site maintaining a list of recently communicated updates. All the sites will need to check this list before accepting new update requests.

## 6. PERFORMANCE COMPARISON

Considering transaction processing steps for a simple case, it is possible to get a view of costs incurred by some of the transaction processing techniques in use. Lindsay [LIND87] considers communication cost, in terms of number of messages, as

the major factor. In addition, there are other factors which affect speed of processing. These factors vary over several orders of magnitudes, for existing techniques, depending on complexity of requests, system load fluctuations, and synchronization conflicts. The overall response time performance for a given technique is determined by three factors[BERN81], which are as follows:

(a) Amount of blocking introduced by the technique;

(b) Number of roll-backs introduced by the technique; and

(c) Amount of local processing overhead introduced.

For comparing communication costs in terms of messages, we examine the locking based approaches to concurrency control. In most cases, more messages are sent concerning locks than are sent to convey data from one site to another. As a result, a coarse granularity is chosen, that is, lockable items are large objects, perhaps even whole relations [ULLM82]. Consider the central node based locking approach [GARC79] [ULLM82]. The central site receives a message, requesting a read-lock. The site sends a message accepting or rejecting the request. In the case of acceptance, another message requires transfer of the items desired from one of the participant sites. Similarly, in the case of the write-request the site running the transaction must send an extra message to the central site, in order to release locks. Figure 6 provides a comparison of the costs incurred, by various

| Conventional Techniques | Write operation | | Read operation | | Operating environment |
|---|---|---|---|---|---|
| | Short messages | Long messages | Short messages | Long messages | |
| Write-locks-all | 2n | n | 1 | 1 | suitable if read dominates |
| Majority | > = 1 + n | n | > = n | 1 | Avoids many dead-locks |
| Primary Site | 2 | n | 1 | 1 | Vulnerable to crash |
| Primary copy Token | 0 to 4M | n | 0 to 4M | 1 | Adaptible to changes in use pattern |
| Central node | 3 | n | 2 | 1 | Vulnerable to crash |

Figure 6 : Cost Comparison of conventional techniques (after [ULL82])

| Proposed Technique | Write operation | | Read operation | | Operating environment |
|---|---|---|---|---|---|
| | Short messages | Long messages | Short messages | Long messages | |
| Priority response transactions | 2 + M | n | 1 | 1 | High priority transactions (but not long) |
| Normal transactions | 1 | n | 1 | 1 | |
| Roll-backed transactions (short) | 2 + M | n | 2 | 2 | Automatic blocking after a reject message |
| Long transactions with blocking | 1 + M | n | 1 | 1 | Transactions accessing large volume of data /or long transactions |

Figure 7 : Cost alternatives in the case of the proposed algorithm

M Number of sites and n Number of copies

locking based methods [ULLM82].

In the case of the proposed technique, the requesting site sends a message to read items. It receives a return message with the values. On completion of the transaction, a request is sent to the coordinator site. The coordinator site performs a validation and sends n messages to the other sites. This is the cost of processing incurred for processing a normal transaction. See Figure 7 for details on various modes of operation adopted by the proposed solution.

The granularity of items used to complete a transaction need not be coarse, in the case of the propsed solution. At the time of a read access the granularity may still be decided on the basis of a trade-off, but while sending a validation request or an update message the granularity may be based on the list of items used to perform the transaction.

Alternatively we compare the cost of processing a transaction with the cost of processing a multisite commit. In order to estimate the cost of processing, we assume that about 5 to 7 percent of the transactions will be in conflict [BADA79]. In addition, there are long transactions, and time-critical transactions that are processed by resorting to blocking. We assume a figure of 15 percent for all such transactions as an extreme, though in practice this figure is likely to be low. We also assume that commit processing requires sending an additional

message. In the case of the blocked transactions, a blocking entry must be sent to all the participating coordinator sites. A total of (0.15n+1) messages are sent on an average in order to process committing a transaction.

Most existing transaction processing methods use two-phase locking and incur additional processing steps in order to commit transactions. A two-phase commit involves an execute message to all sites, response messages from each sites and final commit messages. The total number of such messages is (3n-3). In the case of three phase commit strategy [GARC87], the total overhead is (4n-4) messages. The DDM system [CHAN87] requires a four-phase commit protocol. System SDD-1 [HAMM80] also uses a four-phase approach with each node communicating an additional message to the backup node to safeguard against site failures, during commit processing. An estimated (5n-5) messages need to be exchanged in order to perform a transaction commit in such cases.

The various transaction processing methods are compared in Figure 8. It is seen that the proposed scheme offers less overhead than all existing schemes.

It should be mentioned here that in the proposed scheme, there will be additional local processing overhead for maintaining the CTL, the network status table and the table of read accesses. Most existing schemes also involve overheads for maintaining lock tables and other similar information. For

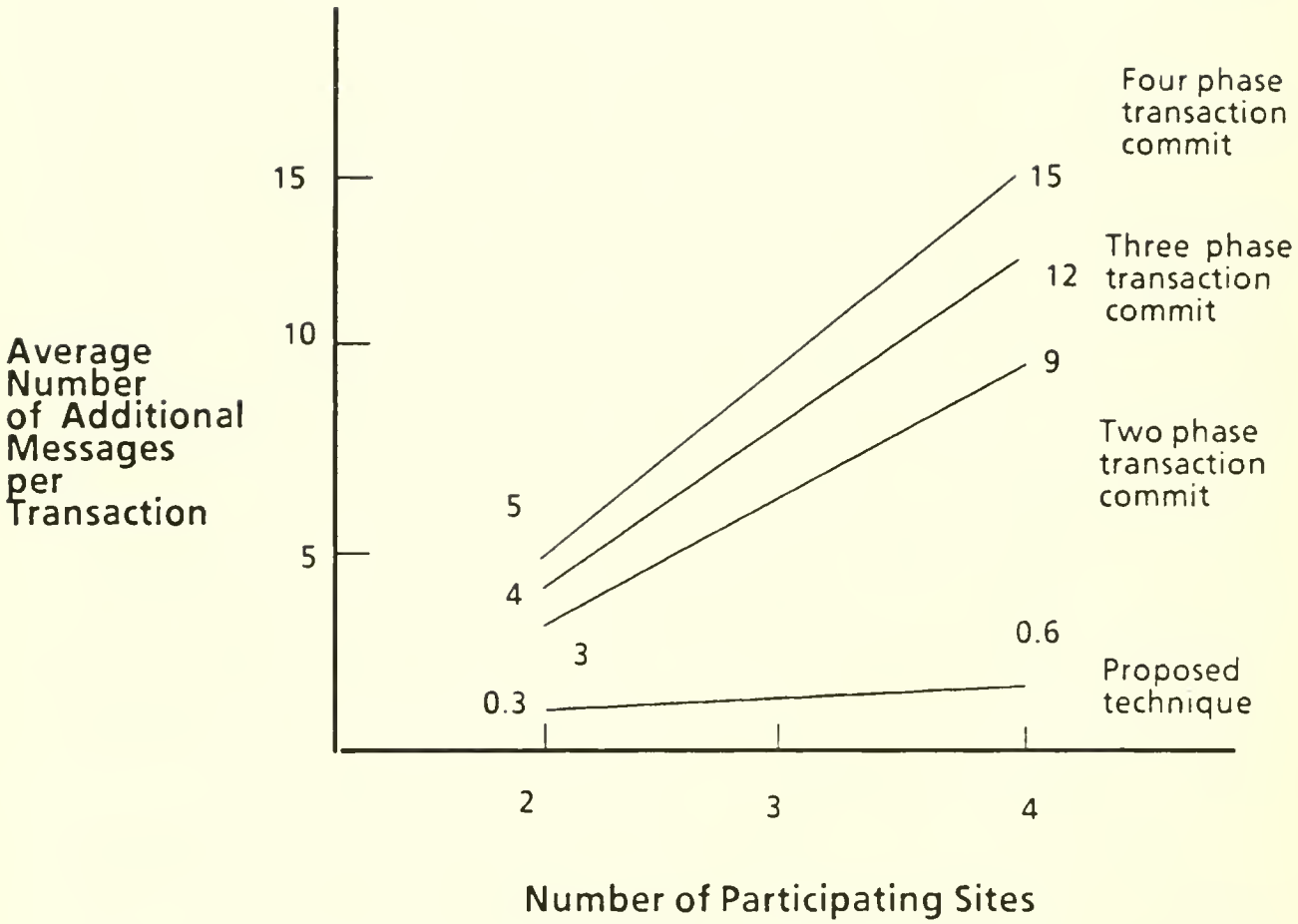


Figure 8: Comparison of two-phase commit processingt cost

example, the DDM system maintains additional information to process read-only transactions.

In terms of transaction blocking, the proposed technique avoids undesirable blocking as each site schedules its own requests on the basis of available information (blocked items by blocking transactions or ongoing transactions). A few conventional techniques, have provision for global deadlock detection. Such services introduce additional delays in transaction processing as locks must be held for some duration of time. These increase the frequency of concurrency conflicts [REDD82a] [GARC87].

The proposed technique also avoids the possibility of repeated roll-backs. Further, unlike the case of conventional transaction processing techniques in which the site failures will cause the dependent transactions to abort, the proposed technique is not subject to such overheads.

In general terms, the proposed scheme offers the following performance advantages over existing schemes:

(a) It reduces the time for which items are blocked due to synchronization or commit processing steps;

(b) It supports multi-site commit and replicated databases without incurring additional cost;

(c) It aims to provide results previously attempted by researchers using multiple versions of a data in a

distributed database; and

(d) It eliminates interference between read-only transactions and update transactions.

Further, the log of transactions generated by the scheme can be examined to determine the optimum database copy allocation strategy among the participating sites.

## 7. FURTHER DISCUSSION

Our technique has been developed on the basis of ideas suggested in [GRAY86] [GRAY81] [GARC87] [BERN83] [KOHL81] [LIND80]. It performs better with fast transfer of information related to updates, items blocked and network status changes. This transfer is achieved easily in the case of LAN based systems (with communication delays of the order of 30 ms), as compared to wide area networks based systems with delays in the range of seconds. However, Gray [GRAY86] feels that such queued message oriented transaction processing will perform well for long transactions, and communications based on wide area networks.

Our approach offers a high degree of autonomy for participating sites, as each site operates independently. Degradation in the performance of a site, or even a site failure will have little effect on the other sites.

Traditionally, centralized processing has been considered more efficient, but it suffers from the problem of limited

safeguard against central site failures. In the context of proposed method of transaction processing, it has been shown to a viable proposition considering that existing methods adopt centralized commit processing and also incur costly transaction-commit procedures [GARC87]. Further efforts can be made to devise distributed transaction processing techniques based on the proposed technique.

The technique has indicated a flexible way of transaction processing that allows synchronization to be performed either by blocking items or by validation at the end of the computation phase. It also permits unblocked processing of read-only transactions and special processing facilities for time-critical transactions.

## 7.1 REPLICATION SUPPORT

Replication of a data at multiple sites implies that updates must be broadcasted to all participating sites. Concurrency control in such a case incurs substantial message costs and delays [GARC82a]. In case any of the replica becomes inaccessible, further updates are inhibited. Using the existing techniques, it is not economical to have a large degree of replication for processing transactions. Our message based transaction processing scheme permits large amount of replication to be supported. In the proposed method, a transaction is

committed by two participents. The coordinator-site communicates the commit message to the requesting site, in order to commit a transaction. Such a mode of transaction processing is supported by [BARB86].

## 7.2 TRANSACTION COMMIT

At the time the transaction is committed through a message from the coordinating node, the requesting site can supply its local clock value to the central site as a part of the acknowledgement message. This value is noted along with the CSN value, as a part of the update message. This time stamp provided by the requesting site can be maintained for each transaction along with other information in transaction logs. Such a log can be used for audit trails, and for answering queries such as "what did the database look like at the end of last month ?", as each committed entry is logged along with the requesting site clock value (also see [SNOD86] [GRAY81]).

# 8. CONCLUSION

Our transaction processing scheme provides more efficient concurrency control and transaction commit capabilities than locking oriented techniques used in existing systems such as SDD-1 [HAMM80], Distributed INGRES [STON77] and System R* [LIND87]. An added advantage of the new scheme is its ability to operate

even when a site fails. Further, the scheme offers the facility for generating a comprehensive record of all databases, as they existed at various points in time, and also for creating transaction logs for audit trails.

Our scheme generates local read access tables managed by participating sites which can be used to provide status information about the executing transactions. In addition to permitting independent processing of read-only transactions and update transactions, our technique permits execution of time-critical transactions on priority basis. While motivating further research in the area of transaction processing based on two site commit method, this scheme offers immediate potential for use in the next generation of distributed database management systems.

## REFERENCES

[ALLC83] Allchin, J.E., and M. S. McKendry, "Synchronization and Recovery of Actions", in Proc. Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, P. Q., Canada, Aug. 83, pp. 31-44.

[ATTA84] Attar, R., Bernstein, P.A., and Goodman, N. "Site Initialization, Recovery, and Backup in a Distributed Database System." IEEE Trans. on Software Engineering, SE-10,No. 6 (Nov. 1984), pp. 645-650.

[BADA79] Badal, D.Z. "Correctness of Concurrency Control and Implementation in Distributed Databases," Proc. COMPSAC Conf., Chicago, Nov. 1979.

[BARB86] Barbara, D. and H. Garcia-Molina, "The Vulnerability of Vote Assignments", ACM Transactions on Computer Systems, Vol. 4, No. 3, Aug. 86, pp. 187-213.

[BAYE80] Bayer, R. Holler, H. and Reiser, A. Parallelism and recovery in Database Systems, Volume 5, June 1980, pp 139-156.

[BERN83] Bernstein, P.A., and Goodman, N. "Multiversion Concurrency Control - Theory and Algorithms", ACM Transactions on Database Systems, Vol. 8, No. 4, Dec. 83, pp. 465-483.

[BERN81] Bernstein, P.A., and Goodman, N. "Fundamental Algorithms for Concurrency Control in Distributed Database Systems." ACM Computing Surveys, Vol. 13,No. 2, June 1981.

[CHAN87] Chan, A., U. Dayal and S. Fox, "An Ada-Compatible Distributed Database Management System", Proceedings of the IEEE, Vol. 75, No. 5, May 87, pp. 674-694.

[CHAN85] Chan, A., and Gray, R., "Implementing Distributed Read-only Transactions", IEEE Transactions on Software Engineering, Vol. SE-11, No. 2, Feb. 85.

[DAVI85] Davidson, S.B., Hector Garcia Molina, and Dale Skeen, "Consistency in Partitioned Networks", Computing Surveys, Vol. 17, No. 3, Sept. 1985.

[DOLE82] Dolev, D. and H.R. Strong, "Polynomial Algorithms for Multiple Processor Agreement", In Proc. of the 14th ACM Symposium on Theory of Computing, San Francisco, May 82, ACM, New York, pp. 401-407.

[DUBO82] Dubourdieu, D.J., "Implementation of Distributed Transactions", in Proc. of Berkeley Workshop on Distributed Data Management and Computer Network, 1982, pp. 81-94.

[ESWA76] Eswaran, K.P., J.N. Gray, R.A. Lorie and I.L. Traiger, "The Notion of Consistency and Predicate Locks in a Database System", Communications of ACM, vol. 19, No. 11, Nov. 76, pp. 624-633.

[GARC82a] Garcia-Molina, H. "Reliability Issues for Fully Replicated Distributed Databases", IEEE Computer, Vol. 16, No. 9, Sept. 82, pp. 34-42.

[GARC87] Garcia-Molina, H. and R.K. Abbott, "Reliable Distributed Database Management", Proceedings of the IEEE, Vol. 75, No. 5, May 87, pp. 601-620.

[GARC82] Garcia-Molina, H. "Elections in a Distributed Computing System." IEEE Trans. on Computers, Vol C-31, No. 1, January 1982.

[GARC81] Garcia-Molina and S. Davidson, "Protocols for Partitioned-Distributed Database Systems", Technical Report, TR-283, Department of Electrical Engineering and Computer Science, Princeton University, March 81.

[GARC79] Garcia-Molina, H., "A Concurrency Control Mechanism for Distributed Databases Which Use Centralized Locking Controllers", Proc. 4th Berkeley Workshop on Distributed Databases and Computer Networks, Aug. 1979.

[GRAY86] Gray, J. "An Approach to Decentralized Computer Systems", IEEE Transactions of Software Engineering, Vol. SE-12, No. 6, June 86, pp. 684-692.

[GRAY81] Gray, J. "The Transaction Concept: Virtues and Limitations." Proc. 7th International Conf. on Very Large Databases, IEEE, Sept. 81.

[GRAY80] Gray, J. "A Transaction Model", Automata Languages and Programming, Sringer Verlag Lecturre Notes in Computer Science, Vol. 80, 1980.

[HAMM80] Hammer, M. and D. Shipman, "Reliability Mechanism for SDD-1: A System for Distributed Databases", ACM Transactions on Database Systems, Vol. 5, No. 4, Dec. 80, pp. 431-466.

[KIM 82] Kim, W., "Auditor: A Framework for Highly Available DB/DC Systems", In Proc. of the 2nd Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, Pa., IEEE Computer Society, Silver Spring, Md., pp. 76-84.

[KOHL81] Kohler, W.H. "A Survey of Techniques of Synchronization and Recovery in Decentralized Computer Systems." ACM Computing Surveys 13,2 (June 1981).

[KUNG81] Kung, H.T., and Robinson, J.T. "On Optimistic Methods for Concurrency Control." ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp. 213-226.

[LAMP81] Lampson, B. "Atomic Transactions", in Lecture Notes in Computer Science, Vol. 105, Distributed Systems : Architecture and Implementation, G. Goos and J. Hartmanis (Eds), Berlin, Springer-Verlag, 1981, pp. 246-265.

[LeLA78] LeLann, G., "Algorithms for Distributed Data-Sharing Systems Which Use Tickets", Proc. of 3rd Berkeley Workshop on Distributed Databases and Computer Networks, Aug. 78.

[LIND87] Lindsay, B.G. "Retrospective of R* : A Distributed Database Management System", Proceedings of the IEEE, Vol. 75, No. 5, May 87, pp. 668-673.

[LIND80] Lindsay, B., and Selinger, P.G., "Site Autonomy Issues in a Distributed Database Management System", IBM Research Report, RJ 2927 (36822), Sept. 80.

[MOHA82] Mohan, C. and A. Silberschatz, "Ditributed Control - Is It Always Desirable", Technical Report, Department of Computer Science, University of Texas at Austin, 1982.

[PAPA84] Papadimitriou, C.M. and P.C. Kanellakis, "On Concurrency Control by Multiple Versions", ACM Transactions on Database Systems, Vol. 9, No. 1, March 84, pp. 89-9

[REDD82a] Reddy, P.G., Bhalla, S., and Prasad, B.E., "A Model of

Concurrency Control in a Distributed Database System", Information Processing Letters, Vol. 14, No. 5, July 1982.

[REDD82b] Reddy, P.G., Bhalla, S., and Prasad, B.E., "Robust, Centralized Certifier Based Concurrency Control For Distributed Databases", Information Processing Letters, Vol. 15, No. 3, Oct. 82.

[REED83] Reed, D.P. "Implementing Atomic Actions on Decetralized Data", ACM Transactions on Computer Systems, Vol. 1, No. 1, Feb. 83, pp. 3-23.

[REED78] Reed, D.P. "Naming and Synchronization in a Decentralized Computer System." Ph.D. dissertation, Tech Report. TR-205, MIT Lab for Computer Science, Sept. 1978.

[RICA81] Ricart, G. and A.K. Agarwala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks", Communications of ACM, Vol. 24, No. 1, pp. 9-17, corrigendum CACM, Vol. 26, No. 2, pp. 147-148.

[ROTH80] Rothnie, J.B., P.A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. Landers, C. Reeve, D. Shipman and E. Wong "Introduction to a System for Distributed Databases (SDD-1)", ACM Transactions on Database Systems, Vol. 5, No. 1, pp. 1-17, March 80.

[SARI86] Sarin, S.K., C.W. Kaufman, and J.E. Sommers, "Using History Information to Process Delayed Database Updates", 12th Intl. Conf. on VLDBs, Kyoto, Aug. 86.

[SCHN79] Schneider, F. "Synchronization in Distributed Programs", Appeared in TOPLAS, Technical Report TR 79-391, Cornell University, 1979.

[SPEC84] Spector, A.Z., et al., "Support for Distributed Transactions in TABS Prototype", Carnegie-Mellon University, Technical Report CMU-CS-84-132, July 84.

[SNOD86] Snodgrass, R. and I. Ahn, "Temporal Databases", Computer, Sept. 86, pp. 35-42.

[STEA81] Stearns, R.E. and Rosenkrantz, D.J. "Distributed Database Concurrency Control Using Before-Values" in Proc. 1981 ACM-SIGMOD Conf. Management of Data, ACM, New York, pp 74-83.

[STON77] Stonebraker, M. and E. Neuhold, "A Distributed database Version of INGRES", in Proc. 2nd Berkeley, Workshop on Distributed Data Management and Computer Networks, Berkeley, CA, May 77, pp. 19-36.

[ULLM82] Ullman, J.D., "Principles of Database Design", McGraw-Hill Publication Co., 1982.

[VIEM82] Viemont, Y.H. and G.J. Gadrin, "A Distributed Concurrency Control Algorithm Based on Transaction Commit Ordering", in Proc. Conf. Fault Tolerant Computer Systems, 1982.

[WEIH87] Weihl, W.E. "Distributed Version Management for Read-only Actions", IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, Jan 87.

[WEIH83] Weihl, W.E. "Data-dependent Concurrency Control and Recovery." Proceedings, 2nd ACM Symposium on Principles of Distributed Computing. 1983.